Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЁТ

По лабораторной работе №10

По курсу: «Функциональное и логическое программирование»

Студент: Керимов А. Ш.

Группа: ИУ7-64Б

Преподаватели: Толпинская Н. Б.,
Строганов Ю. В.

Москва

2020

Практическая часть

Задание 1. Пусть `list-of-list` список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов `list-of-list`, т. е. например для аргумента `((1 2) (3 4))` -> `4`.

```
(defun sum-of-lengths-process (list-of-list acc)
  (if list-of-list
      (sum-of-lengths-process
        (cdr list-of-list)
        (+ acc (length (car list-of-list))))
      acc))

(defun sum-of-lengths (list-of-list)
  (sum-of-lengths-process list-of-list 0))
```

Функция `sum-of-lengths-process` принимает список списков `list-of-list` и накопленную сумму `acc`. Если дошли до конца списка, возвращает накопленную сумму. Иначе, вычисляется рекурсивно для хвоста и накопленной суммы, к которой прибавлена длина списка в голове `list-of-list`.

Примеры работы:

list-of-list	**(sum-of-lengths list-of-list)**
`((1 2) (3 4))`	`4`
`((1 2 (3 3 3)) (4 5 (6 6 6)))`	`6`

Задание 2. Написать рекурсивную версию (с именем `reg-add`) вычисления суммы чисел заданного списка. Например: `(reg-add (2 4 6))` -> `12`.

```
(defun reg-add-process (lst acc)
  (if lst
      (reg-add-process
        (cdr lst)
        (+ acc (car lst)))
      acc))

(defun reg-add (lst)
  (reg-add-process lst 0))
```

Функция `reg-add-process` принимает список чисел `lst` и накопленную сумму `acc`. Если дошли до конца списка, возвращает накопленную сумму. Иначе, вычисляется рекурсивно для хвоста и накопленной суммы, к которой прибавили число в голове списка.

Примеры работы:

lst	**(reg-add lst)**
`(1 2 3 4)`	`10`
`(1 2 3 4 5)`	`15`
`()`	`0`

Задание 3. Написать рекурсивную версию с именем `recnth` функции `nth`.

```
(defun recnth (n lst)
  (cond ((null lst) nil)
        ((= n 0) (car lst))
        (t (recnth (- n 1) (cdr lst)))))
```

Функция `recnth` принимает индекс `n` и список `lst`. Если дошли до конца списка, возвращает `nil`. Иначе, если $n = 0$, возвращает голову списка. Иначе, вычисляется рекурсивно для $n - 1$ и хвоста списка.

Примеры работы:

n	lst	(recnth n lst)
2	(1 2 3 4)	3
4	(1 2 3 4)	nil
4	()	nil

Задание 4. Написать рекурсивную функцию `alloddr`, которая возвращает `t` когда все элементы списка нечетные.

```
(defun alloddr-process (lst acc)
  (if (or (null acc)
          (null lst))
      acc
      (alloddr-process
        (cdr lst)
        (oddp (car lst)))))

(defun alloddr (lst)
  (alloddr-process lst t))
```

Функция `alloddr-process` принимает список чисел `lst` и результат `acc`. Если дошли до конца списка или результат `nil`, возвращает результат. Иначе, вычисляется рекурсивно для хвоста и результата, который равен `t`, если голова списка является нечётным числом, и `nil` иначе.

Примеры работы:

lst	(alloddr lst)
(1 2 3 4 5)	nil
(1 3 5)	t
(1 3 5 4)	nil
()	nil

Задание 5. Написать рекурсивную функцию, относящуюся к хвостовой рекурсии с одним тестом завершения, которая возвращает последний элемент списка — аргументы.

```
(defun rec-last (lst)
  (if (cdr lst)
      (rec-last (cdr lst))
      (car lst)))
```

Функция `rec-last` принимает список `lst`. Если хвост списка не пустой, функция вычисляется рекурсивно для него (хвоста списка). Иначе, возвращает голову списка (последний элемент).

Примеры работы:

lst	(rec-last lst)
(1 2 3 4)	4
(1 2 3 4 5)	5
()	nil

Задание 6. Написать рекурсивную функцию, относящуюся к дополняемой рекурсии с одним тестом завершения, которая вычисляет сумму всех чисел от 0 до n-ого аргумента функции. Вариант: 1) от n-аргумента функции до последнего $>= 0$.

```
(defun sum-to-n (lst n)
  (if (and lst (> n 0))
      (+ (car lst)
         (sum-to-n (cdr lst) (- n 1)))
      0))

(defun sum-from-n (lst n)
  (if lst
      (+
        (if (<= n 0) (car lst) 0)
        (sum-from-n (cdr lst) (- n 1)))
      0))
```

Функция `sum-to-n` принимает список `lst`. Если список не пустой и $n > 0$, возвращает сумму головы списка с результатом рекурсивного вычисления функции для хвоста списка и $n - 1$. Иначе, возвращает 0.

Функция `sum-from-n` принимает список `lst`. Если список пустой, возвращает 0. Иначе, возвращает сумму, где первое слагаемое является головой списка в случае, если $n < 0$, и нулём иначе, а второе слагаемое — результатом рекурсивного вычисления функции для хвоста списка и $n - 1$.

Примеры работы:

lst	n	(sum-to-n lst n)	(sum-from-n lst n)
(1 2 3 4 5)	2	3	12
(1 2 3 4 5)	5	15	0
(1 2 3 4 5)	0	0	15

Задание 7. Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка, возможно создавая некоторые вспомогательные функции.

```
(defun last-odd-process (lst acc)
  (if lst
      (last-odd-process
        (cdr lst)
        (if (oddp (car lst))
            (car lst)
            acc))
      acc))

(defun last-odd (lst)
  (last-odd-process lst nil))
```

Функция `last-odd-process` принимает список `lst` и результат `acc`. Если дошли до конца списка, возвращает результат. Иначе, вычисляется рекурсивно для хвоста и результата, который равен голове списка, если она (голова списка) является нечётным числом, или прежнему результату иначе.

Примеры работы:

lst	(last-odd lst)
(1 2 3 4)	3
(1 2 3 4 5)	5
(2 4 6)	nil
()	nil

Задание 8. Используя cons-дополняемую рекурсию с одним тестом завершения, написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

```
(defun sqr (number)
  (* number number))

(defun sqr-lst (lst)
  (if lst
      (cons (sqr (car lst))
            (sqr-lst (cdr lst)))
      nil))
```

Функция sqr-lst принимает список lst. Если список не пустой и возвращает список, голова которого является квадратом исходной головы списка, а хвост — результатом рекурсивного вычисления функции для хвоста исходного списка. Иначе, возвращает nil.

Примеры работы:

lst	(last-odd lst)
(1 2 3 4)	(1 4 9 16)
()	()

Задание 9. Написать функцию с именем select-odd, которая из заданного списка выбирает все нечетные числа. (Вариант 1: select-even, вариант 2: вычисляет сумму всех нечетных чисел (sum-all-odd) или сумму всех четных чисел (sum-all-even) из заданного списка)

```
(defun select-odd (lst)
  (if lst
      (if (oddp (car lst))
          (cons (car lst)
                (select-odd (cdr lst)))
          (select-odd (cdr lst)))
      nil))

(defun select-even (lst)
  (if lst
      (if (evenp (car lst))
          (cons (car lst)
                (select-even (cdr lst)))
          (select-even (cdr lst)))
      nil))

(defun sum-all-odd-process (lst acc)
  (if lst
      (sum-all-odd-process
        (cdr lst)
        (if (oddp (car lst))
```

```
            (+ acc (car lst))
            acc))
      acc))

(defun sum-all-odd (lst)
  (sum-all-odd-process lst 0))

(defun sum-all-even-process (lst acc)
  (if lst
      (sum-all-even-process
        (cdr lst)
        (if (evenp (car lst))
            (+ acc (car lst))
            acc))
      acc))

(defun sum-all-even (lst)
  (sum-all-even-process lst 0))
```

Примеры работы:

lst	(select-odd lst)	(select-even lst)	(sum-all-odd lst)	(sum-all-even lst)
(1 2 3 4 5)	(1 3 5)	(2 4)	9	6
(1 3 5)	(1 3 5)	()	9	0
()	()	()	0	0

Теоретическая часть

Способы организации повторных вычислений в Lisp

Для организации повторных вычислений в Lisp могут быть использованы функционалы и рекурсия.

Что такое рекурсия? Классификация рекурсивных функций в Lisp

Рекурсия — это ссылка на определяемый объект во время его определения.
В Lisp существует классификация рекурсивных функций:

- простая рекурсия — один рекурсивный вызов в теле;
- рекурсия первого порядка — рекурсивный вызов встречается несколько раз;
- взаимная рекурсия — используется несколько функций, рекурсивно вызывающих друг друга.

Различные способы организации рекурсивных функций и порядок их реализации

При организации рекурсии можно использовать как функции с именем, так и локально определенные с помощью лямбда-выражений функции. Кроме этого, при организации рекурсии можно использовать функционалы или использовать рекурсивную функцию внутри

функционала. При изучении рекурсии рекомендуется организовывать и отлаживать реализацию отдельных подзадач исходной задачи, обращая внимание на эффективность реализации и качество работы, а потом, при необходимости, встраивать эти функции в более крупные, возможно в виде лямбда-выражений.

Способы повышения эффективности реализации рекурсии

- **Хвостовая рекурсия**. В целях повышения эффективности рекурсивных функций рекомендуется формировать результат не на выходе из рекурсии, а на входе в рекурсию, все действия выполняя до ухода на следующий шаг рекурсии.

  Преобразование не хвостовой рекурсии в хвостовую, возможно путем использования дополнительных параметров. В этом случае необходимо использовать функцию-оболочку для запуска рекурсивной функции с начальными значениями дополнительных параметров.

- **Дополняемая рекурсия**. При обращении к рекурсивной функции используется дополнительная функция не в аргументе вызова, а вне его.

- **Выделяют группу функций множественной рекурсии**. На одной ветке происходит сразу несколько рекурсивных вызовов. Количество условий выхода также может зависеть от задачи.